Statistical Measures as Predictors of Compression Savings

Senior Honors Thesis to fulfill the requirements of Graduation with Distinction

By William Culhane

The Ohio State University
May 2008

Project Advisor: Professor Bruce Weide, Department of Computer Science and Engineering

Abstract

Different compression algorithms run on the same file will result in differently sized compressed files. Choosing the algorithm which will give the smallest resulting file has some advantages. It would be useful to be able to do this without having to take the resources to run every algorithm in question. In order to do this, some predictive measure has to be found.

Chapter 1 – Introduction

1.1 Problem and Application

There are certain times when transmission of data is expensive and some extra computational time to minimize transmission could be a good investment. This could be because of energy costs, such as transmitting over a large area with little infrastructure, as exists in space travel. It could also be useful when computational speed is much more than transmission speed. This problem will become more important as processors get more powerful, but bandwidth remains limited.

The purpose of this project is to find a set of statistical measures in the byte representation of a file which predicts the compressibility of the file via different compression algorithms, allowing a quick pass to be made over the file to determine which, if any, algorithm to use.

A highly predictive set of measures needs to be found, making it possible to estimate the compressed file size without running the compression algorithms. This allows for time and memory savings over running multiple algorithms over the same set of data while still giving the smallest resulting file. The measures need to be able to predict the compression of various algorithms accurately enough to predict with some accuracy which algorithm to run.

In order to be viable, the measure also has to have some advantage over compressing the file with the different algorithms and simply choosing the smallest file. This will sometimes be a time savings. The measure will be computationally light enough that it will take less time than running multiple algorithms. It could also be memory savings. With a single pass algorithm, the whole resulting file would need to be saved for size comparison. Savings of double pass algorithms may be able to be predicted after the first pass without using up a lot of extra memory.

1.2 Compression Overview

Compression algorithms are forms of data transformation which normally result in a smaller file. There are four types of compression algorithms (Salomon, Introduction). For this project, two different types of algorithms were chosen. Huffman, a statistical algorithm, and variations on Lempel-Ziv-Welch (LZW), which is a dictionary based method. As well as representing different types of algorithms, these are two of the most common forms of compression (Buchanan, B2.6).

1.3 Related Work

There have been some studies as to which compression algorithms are better for different files. Some of them, such as the ones displayed at the Maximum Compression website ("Lossless"), focus on pigeon holing file types by the outside information of what sort of information they contain. Common file extensions are listed separately and grouped by

categories including text files, graphics files, and executable files. All of the effort was based on mapping known file types to their best compression algorithm. There is not a concerted effort to find trends within files independent from what the file represents. There is no analysis of the files binary or bit composition in relation to how it may affect the compression.

Other studies which study different algorithms do not give any indication of the content of the files used for the study. In one paper found, the files were only referred to by labels given by the researchers (Hayashi et al). It was not a focus of the paper what aspects of the file may have contributed to the difference in compression of the algorithms. Differences were accredited entirely to algorithms.

More searching reveals more metrics for choosing how to group source data. The variation in source data uncovered in research indicates that researchers determine their own source for each experiment. No single set of files was found as a standard set to test compression algorithms that would lend this project extra credibility.

There has also been some study on the most compressed form of a file. Claude Shannon defined a relationship between the entropy of data and the maximum compression that can be achieved on that data (Acharya, 3.2). His work states that the informational content of a symbol used to represent something in a data stream is connected to the frequency of that symbol's use. Lesser used symbols contain more information than common ones because their appearance is less expected and represent a break from the

norm. By analyzing the entropy of a file, it is possible to provide a bound for the amount of compression that can be achieved by providing a measure of the amount of redundant data that can be removed from that file during compression. While this is meant to give a gauge of the effectiveness of an algorithm, it could also prove to be a good starting point for a predictive measure.

1.4 Thesis

There are two parts to the thesis of this work. The first part is that there are statistical measures in the data or files being compressed which can predict the savings that various compression algorithms will have. By doing minimal work, it will be possible to determine how much smaller the file can be made with some accuracy.

The second part is relies on the successful completion of the first part. Once measures are found which predict the savings gained by different algorithms, it will be possible to figure out which algorithm would be best to compress any particular file without needing to compress it. This part of project is checking to see if the equations found in the first part are sufficient for deciding on an algorithm. It has to be checked that the measures are predictive enough to be used.

1.5 Organization of the Paper

Chapter two contains information on the setup and implementation of the project. It explains the lead up and execution of the experiment including assumptions, algorithm details, and measures. Following that are the results. Those for the predictive ability of the measure are in chapter three. Those for the direct comparisons between algorithms are in chapter four. Chapter five contains conclusions and possibilities for future work.

Chapter 2 – Test Parameters

2.1 Data Source

One experimental variable is the source data. Other projects which measure compression use their own test files. In some cases, they choose files which contain particular traits (Hayashi et al.). Others display their test files so others may replicate their work (Gilchrist), ("Lossless"). The result is that benchmark files vary from source to source, and there is no standard benchmark that would allow general conclusions for any given data set.

For this experiment, a hard drive from a three-year-old desktop computer was used. Files with the same extension were grouped and traversed in such a way that the end result was the equivalent of concatenating all files with the same extension and treating that as a single file. This was done based on the assumption that files of the same type had similar traits. Each extension makes up a single data point in the final data. Data points with a total size of less than 5 kilobytes were ignored because the savings of the 12-bit LZW algorithm is not consistent when the data size was too small. These points were removed from the other analyses for consistency. The files were traversed at the byte level with no regard for structure or organization of the file type because this project is meant to find trends which could be applied to predicting compression of any data in the future. It was assumed that things like header, footer, and segment layouts were not as important as the

byte distribution. This gives a large number of test points to allow the conclusions to be expanded to many types of files, including those not included in the analysis, as long as a trend is shown to exist in the data.

2.2 Algorithms

2.2.1 LZW

For testing dictionary algorithms, two forms of LZW were used. Initially, a very simple version of LZW was used (Salomon, 4.3.12). A table was initialized by entering the bit strings "1" and "0" with corresponding integer values 1 and 0 respectively. The data was read from the beginning until a string of bits was found which was not in the table. That string was entered into the table paired with an integer equal to the size of the table prior to the new string's being entered. The substring of the entered string which does not contain the final bit was looked up in the table, and that substring was replaced with the corresponding integer value. The process was restarted with the final bit of the original string, which was not replaced with the integer. This was continued until the table was filled. With 8-bit LZW, that happened after 256 entries; with 12-bit LZW, after 4096. After the final table size was realized, new entries were not added to the table, which the book explains is the simplest way of handling the full table (Salomon, 4.3.8). The data was still traversed in the same manner to find the largest string already in the table. They were replaced with the corresponding integers. This was done until the end of the data was reached.

An example of LZW with a table size of 8 follows. If the original file were 111101010110001 and the table were initialized to {(“0”, 0), (“1”, 1)}, the replacement would result in a compressed file containing the integer for the first replacement. The replaced bit would be removed from the buffer still to be replaced, and the table would add an entry for the replaced bit plus the next bit in the buffer. The result would be 1, 11101010110001, {(“0”, 0), (“1”, 1), (“11”, 3)}. Another step would give 1-3, 101010110001, {(“0”, 0), (“1”, 1), (“11”, 3), (“111”, 4)}. The next step gives 1-3-1, 01010110001, {(“0”, 0), (“1”, 1), (“11”, 3), (“111”, 4), (“10”, 5)}. One more step results in 1-3-1-0, 1010110001, {(“0”, 0), (“1”, 1), (“11”, 3), (“111”, 4), (“10”, 5), (“01”, 6)}. Another step results in 1-3-1-0-5, 10110001, {(“0”, 0), (“1”, 1), (“11”, 3), (“111”, 4), (“10”, 5), (“01”, 6), (“101”, 7)}. At this point, the table is full, so replacements are made without adding to the table. The final file is 1-3-1-0-5-7-5-0-0-1. These are integer values which will all be represented by the number of bits required to represent the maximum size of the table. In this case, there would be a final file size of 30 bits, which is larger than the original file of 15 bits. In this case, the longest string replaced was three bits. If a four bit string had made it into the table, every time it would have been replaced it would have saved a bit. A more visual representation with different data is shown in Example 1 below.

Initial Table = { (0, “0”), (1, “1”) } Initial File = 011110100000101110101101
Step 1: 01 is the first string not in the table. All but final bit of that string are removed, which is shown with the strikethrough effect. The integer mapped to the longest string in the table is placed in the compressed file. Table = { (0, “0”), (1, “1”), (2, “01”) } File = ~~0~~11110100000101110101101
Step 2: Repeat step 1 process for 11 Table = { (0, “0”), (1, “1”), (2, “01”), (3, “11”) } File = ~~01~~11110100000101110101101
Step 3: Table = { (0, “0”), (1, “1”), (2, “01”), (3, “11”), (4, “111”) } File = ~~0111~~1010000101110101101
Step 4: Table = { (0, “0”), (1, “1”), (2, “01”), (3, “11”), (4, “111”), (5, “10”) } File = ~~01111~~010000101110101101 New File = 0 1 3 1 2 . . .

Example 1: A partial example of LZW abstracted beyond the bit level to the integer level.

There is a technique called variable length code LZW compression which adds more savings to this technique (Buchanan, B6.6). It takes advantage of the fact that when the table size is two, only one bit is needed to represent the integer. For table sizes three and four, only two bits are needed. As the table grows, the number of bits required to

represent it grows every time a new power of two is passed. For this part of the experiment, the algorithm did not take advantage of this fact. This is because it offers a fixed amount of savings as long as the table is filled. It offers no additional savings after that. For 12-bit LZW compression, 4083 bits can be saved. In the above 3-bit LZW, it would have saved 4 bits. This is a fixed amount, and it would have skewed the savings percentages of the smaller data points.

For the second version of LZW, variable length coding was employed. 12-bit LZW was basically implemented again, but the table was not frozen when it was filled. It was completely cleared and (“0”, 0) and (“1”, 1) were reinserted, as was suggested in Salomon’s book as an alternative to freezing the table (Salomon, 4.3.8). This allowed those 4083 bits to be saved multiple times. It also allowed the algorithm to structure the compression on smaller segments of the data instead of using only the beginning of the data to sample which strings should be in the table, which had been the effective result of continuing to use a filled table without inserting new pairs. The example for 3-bit LZW can be adjusted to show this by clearing the table when it is filled and restarting the table filling at that point. The compressed version of the file and the buffer to be compressed would stay the same when the table is cleared. Each integer in the compressed file would not be replaced by three bits, but by the number of bits required to represent the size of the table at the time of the replacement.

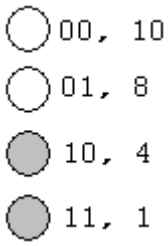
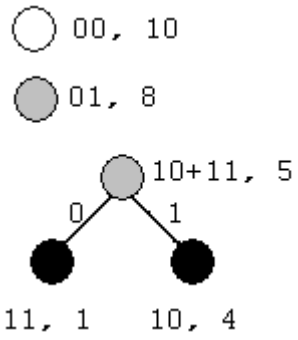
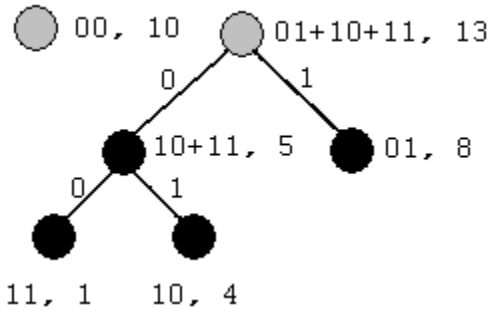
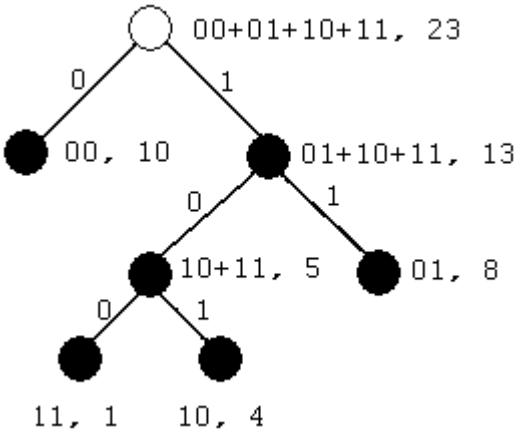
2.2.2 Huffman

The statistical algorithm tested is a basic form of Huffman encoding as explained by Salomon (3.2.8). Special effort to minimize the size of the variance as explained by Salomon is not taken because buffer size is not a factor in the experiment. Each byte is paired with the occurrences of itself in the file. An empty string is associated with each byte in a separate structure. The two nodes with the smallest number of total bytes are paired. A 0 is prepended to the string associated with the nodes in the smaller pair, and a 1 is prepended to the strings of those in the larger. A new node is created with the total number of occurrences equal to the sum of the two combined nodes, and the two nodes that were combined are removed. The process is repeated until only one node remains. Each byte is represented by a unique bit string in the second structure. More common bytes get associated with shorter strings and less common ones with longer strings.

When the bytes are replaced with their strings, the resulting data are usually shorter than the original. In order to decompress the data, the mapping table is required. This would go before the encoded data. The size of the table can vary. If it is represented as 256 consecutive entries of an eight bit integer stating the size of the string and the bit string representing the byte corresponding to that entry, the size of the table can be between 4096 and 34,688 bytes inclusive, with a larger table corresponding to a table of more skewed data. Based on the assumption that this will only be used for larger datasets, this is a relatively small amount. The size of that table is not incorporated with this experiment when determining savings because the size would have skewed the smaller

data points by lessening their predicted savings percentage. When the size of the file is sufficiently large, this addition of size is negligible, but it would have resulted in a smaller savings percent recorded for data points representing files which were not sufficiently large enough to mask this added size.

The following is an example of Huffman with two bit strings instead of bytes. The original data set could be {(00, 10), (01, 8), (10, 4), (11, 1)}. The first entry in each pair represents the string in the original file. The second entry is the number of occurrences of that string. The smallest two nodes are those with 11 and 10. They are combined in the tree, and their corresponding strings are prepended with the appropriate bit. The result is a set of nodes to still be combined consisting of {(00, 10), (01, 8), (10 + 11, 5)} and the map of the strings with {(00, “”), (01, “”), (10, “1”), (11, “0”)}. Another step would result in {(00, 20), (01 + 10 + 11, 13)} and {(00, “”), (01, “1”), (10, “01”), (11, “00”)}. The final step would result in {(00 + 01 + 10 + 11, 23)} and {(00, “0”), (01, “11”), (10, “101”), (11, “100”)}. A visual representation of this can be seen in Example 2 below.

<table>
<tr>
<td>1.
 
</td>
<td>2.
 
</td>
</tr>
<tr>
<td>3.
 
</td>
<td>4.
 
</td>
</tr>
<tr>
<td colspan="2">In each step, the two smallest nodes remaining are colored gray. The nodes which have already been combined are black.
In order to get the replacement strings for the original, one can traverse the trees and read the caption of each branch on the way down.</td>
</tr>
</table>

Example 2: Huffman Encoding

The saving in bits before the addition of the table is the sum of length of each new string multiplied by the occurrences of the corresponding occurrences in the original file subtracted from the length of the original string multiplied by the sum of all occurrences.

In this case, that would be 2*(10 + 8 + 4 + 1) - (1*10 + 2*8 + 3*4 + 3*1), which is 5 bits of savings on a file of 46 bits. If the original file had been perfectly evenly distributed, the resulting combination of nodes would have resulted in bit string all equal in length to the original bit strings. This would have resulted in no savings, even before the size of the table was taken into consideration.

2.3 Possible measures

Three statistical measures are studied in an attempt to predict the savings of the various algorithms. The three measures tracked are standard deviation on the bytes, standard deviation of the difference of consecutive bytes, and standard deviation of the XORed value of consecutive bytes. These were chosen because it was believed that the evenness of the distributions may reveal patterns within the data which could predict compressibility in line with Shannon's work. Each shows a different pattern in the file. The standard deviation of the distribution of the bytes shows the level of evenness of the bytes of the file. The other two measure the level of evenness of the order of the bytes. It was expected that these patterns within the file would predict the spread of the file and therefore its compressibility.

In each case, the occurrence of each value in the distribution is tracked. For the bytes and the XORed bytes, there are 256 possible values. For differences, there are 513. The occurrences are then divided by the total number of entries in the distribution. The standard deviation is calculated by Formula 1 below.

$$\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(x_i-\frac{1}{N}\right)^2} \quad (1)$$

Standard deviation- N is the number of possible values and x_i is the ratio of the occurrences of that reading over the total number of readings.

2.4 Assumptions

Several assumptions are made that can be justified given the expected use of the project's results. The first assumption is that files should be rather large — at least several megabytes. This allows certain simplifications to be made when tabulating savings, which will be explained further later in this section. In order to get sufficiently large data points for the data to match this assumption, it is assumed that files with the same extension have similar traits. This also leads to the assumption that trends in the data point as a whole are more important than the shape of a single file. Another assumption is that the data will not change after being read. That allows two pass algorithms to be examined as well as single pass.

The choice of the hard drive and file concatenation rely heavily on a couple of the assumptions made for the experiment. The first is that files of the same extension have similar traits. This is not a claim that those files are identical. It is a statement that files with the same extension have relatively similar trends in distributions. They will more often be more like each other in this manner than randomly chosen files with different

extensions. This will be verified if there is a good spread in the final data. If they have dissimilar traits, they will cancel to an extent, and things would group around the middle and there would be very few points at the ends. The other assumption is that the trends in the file are more important than the shape of the file. Both of these assumptions will be verified if the majority of the final data conforms to the trends found. There will be some files extensions that are relatively rare and will test that file shape has little impact on compression if they compare similarly to common extensions. If there are a handful of points which stick out, it is possible that the unique files that do not have concatenation are forming their own trends that are different than those created by the concatenated ones. This would disprove the assumption that byte distribution is more important than the structure of the file.

Another assumption is added to the experiment by this choice in source data, and it can also be verified by the final data. This choice assumes that the files on the hard drive have enough variety that the results can be generalized to an extent to data other than that on the hard drive, including specialized data files. If the spread of the final data is large enough, it will show that the hard drive had enough variety that the results may be extrapolated to an extent to other data file types.

Chapter 3 – Measures as Predictors

The first part of this experiment is focused on finding statistical measures which predict the savings achieved by the algorithms. This is required to be able to be able to choose the algorithm which will offer the greatest compression on a file.

3.1 Huffman

For Huffman Encoding, it is known that this has greater compression for data with a more skewed distribution of the length of the replacement strings (Crowcroft et al, 4.5.2). Therefore, the standard deviation of the distribution of the bytes is used. The resulting graph can be seen in Figure 1 below. The savings percentage is taken to the .21 power in order to make the result more logarithmic so a better predictive line of fit can be determined. The coefficient of determination of the resulting equations shows that nearly 93% of the logarithmic variation on the vertical axis can be reasonably explained by the variation on the horizontal axis. This means that the changes in the standard deviation of the distribution of the bytes can reasonably explain 93% of the change in a function of the savings achieved. That shows a very strong correlation to Equation 2, shown below the figure. The line of best fit appears to overestimate the amount of savings near the right side of the graph, which could be important when using this as a predictor for which algorithm to use. The other measures explained in Equation 1 above are not explored because patterns between bytes should have no bearing on the creation of the tree. This is

because the tree is formed based on the rate of occurrences of bytes, and is not affected by the order of the bytes.

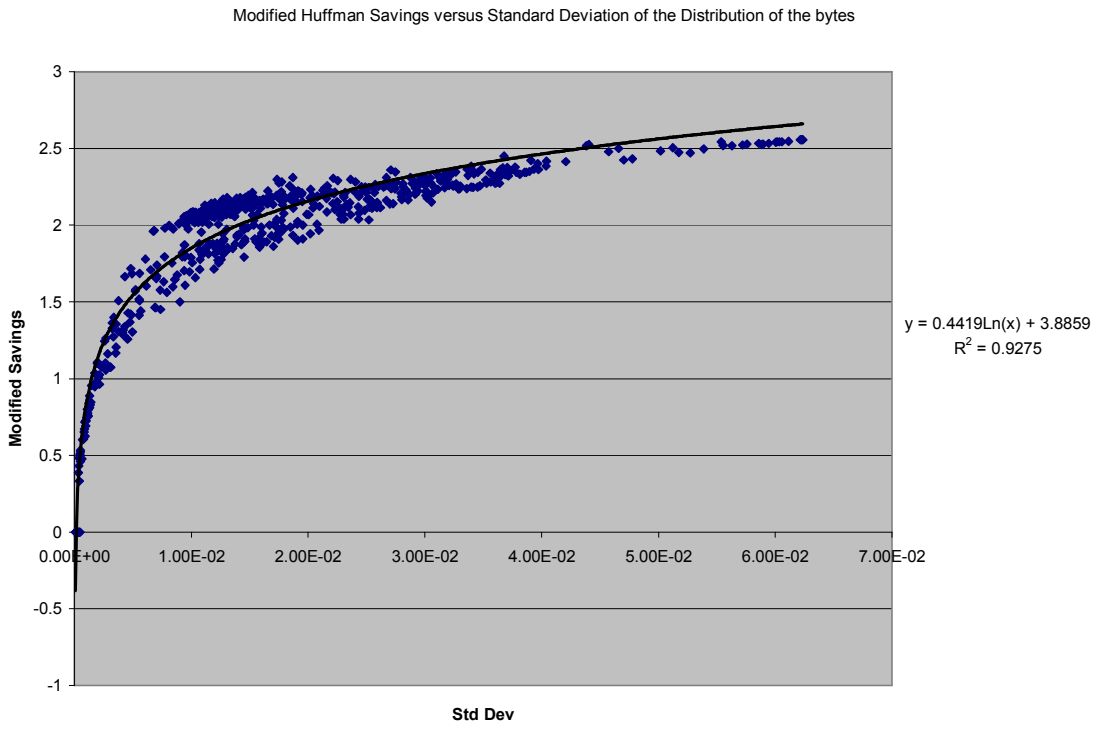


Figure 1: Graph showing the correlation between the standard deviation of the distribution of the bytes and the percentage savings using Huffman encoding taken to the .21 power.

$$\mathbf{F(x)} = \left(.4419\ln(x) + 3.8859\right)^{4.762} \qquad (2)$$

Logarithmic line of best fit for Figure 1. *x* stands for the standard deviation of the distribution of the bytes and the result is the expected percent savings.

3.2 12-bit LZW

12-bit LZW savings are tracked against all three measures explained above. The resulting graphs can be seen below in Figures 2 and 3. The XORed distribution standard deviations and difference standard deviations correlate fairly well to the amount of savings. The coefficients of correlation for a linear relationship are .72 and .75. 12-bit is then graphed against the standard deviation of the distribution of the bytes in Figure 4. The coefficient of correlation for a linear relationship is .76, which is higher than those of the other comparisons. In fact, by removing the biggest outlier of (5.95E-02, -47.55), the coefficient is raised to almost .81. The lines of best fit for these two cases are shown in Equations 2 and 3. The lines of best fit for the other two measures are not shown because they do not correlate as well to the savings.

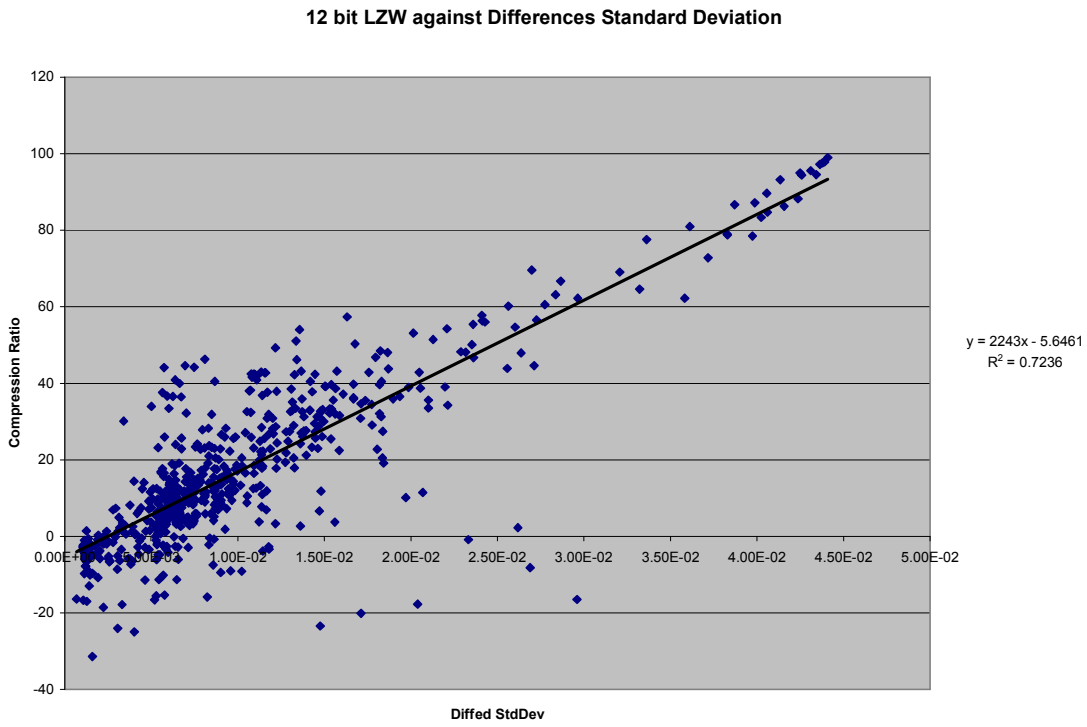


Figure 2: Graph showing the correlation between the standard deviation of the distribution of the difference of consecutive bytes and the percentage savings using 12-bit LZW.

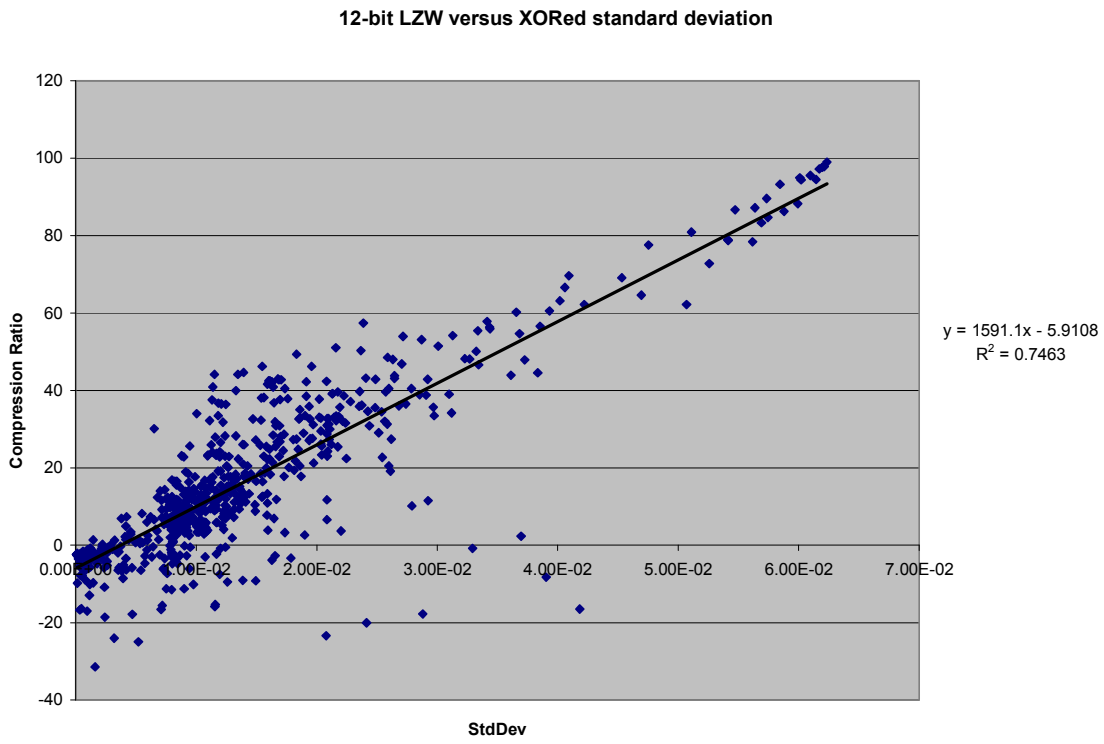


Figure 3: Graph showing the correlation between the standard deviation of the distribution of the XORed values of consecutive bytes and the percentage savings using 12-bit LZW.

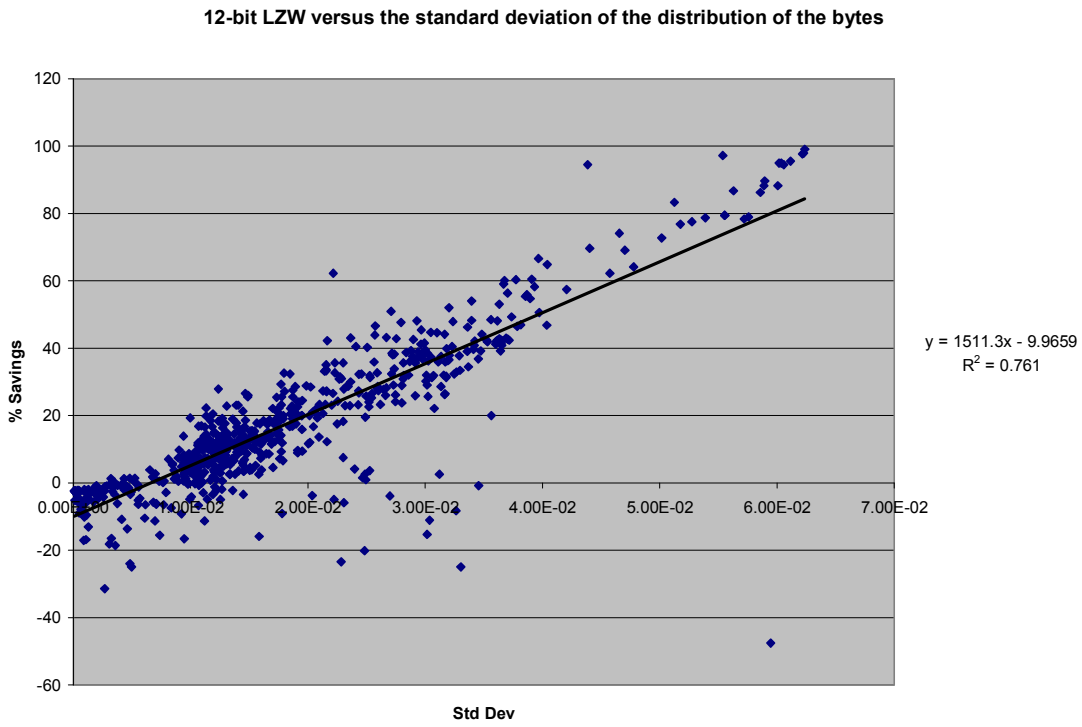


Figure 4: Graph showing the correlation between the standard deviation of the distribution of the bytes and the percentage savings using 12-bit LZW.

$$\mathbf{F(x)} = 1511.3x - 9.9659 \qquad (3)$$

Linear line of best fit for Figure 4. x is the standard deviation of the distribution of the bytes.

$$\mathbf{F(x)} = 1561.6x - 10.68 \qquad (4)$$

Linear line of best fit for Figure 4 with the largest outlier removed. x is the standard deviation of the distribution of the bytes.

3.3 Variable length LZW

Because of the greater correlation of 12-bit LZW to the standard deviation of the distribution of the bytes than either of the other two measures, that was the only variable used as a predicting trend with variable LZW. The resulting graph was placed in Figure 5 below. The coefficient of correlation says that over 92.5% of the linear change in the savings percentage of the algorithm can be explained by the change in the standard deviation of the distribution of the bytes. The line of best fit which matches this coefficient of correlation is shown in Equation 5. Another strength of this algorithm is the lack of obvious outliers below the trend line such as can be observed in the 12-bit LZW analysis in Figure 4 above.

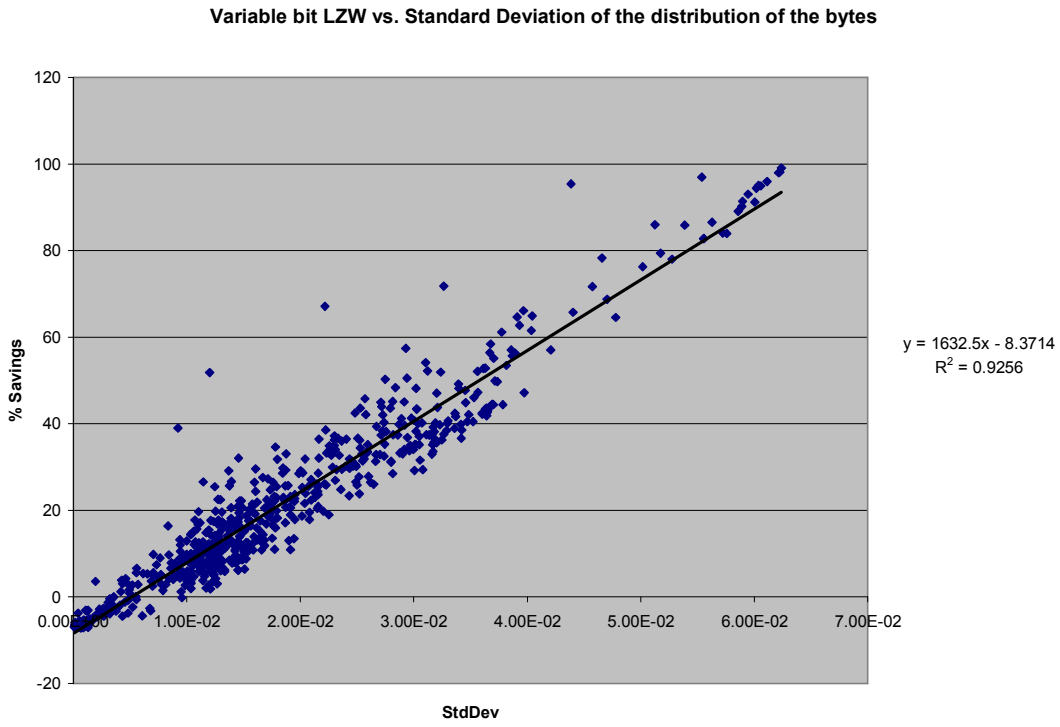


Figure 5: Graph showing the correlation between the standard deviation of the distribution of the bytes and the percentage savings using variable LZW.

$$\mathbf{F(x)} = 1632.5x - 8.3714 \qquad (5)$$

The linear line of best fit for Figure 5. x is the standard deviation of the distribution of the bytes.

Chapter 4 – Direct Comparisons

An important aspect of this project is choosing when to use which algorithm, which means that it has to be checked if the predictors that were found can actually be used for their intended purpose. The following analyses use the differences in savings percentages on each data point to find a predictive trend for when one algorithm should be used over another. The trends were created using the standard deviation of the distribution of the bytes as the independent variable because of that measure's ability to predict the compression savings of all three algorithms.

4.1 Huffman against LZW

Figure 6 below shows a comparison of savings using Huffman Encoding and 12-bit LZW. Each point is the difference in savings achieved by the algorithms for an individual file extension. Points above the horizontal axis represent points that had better compression savings by Huffman than by 12-bit LZW. Based on Equations 2 and 4 above, Huffman should be better until the standard deviation of the distribution of the bytes is about .12. Figure 6 supports the dominance of Huffman for smaller standard deviations of the distribution of the bytes. Near the upper edge of the data collected, 12-bit LZW does better against Huffman than lower in the data. However, it never quite completely appears to take over as the better compression algorithm except for when Huffman hits its ceiling savings percentage. This occurs when every byte is replaced by

the smallest possible replacement string of 1 bit, which would correlate to a savings of 87.5%.

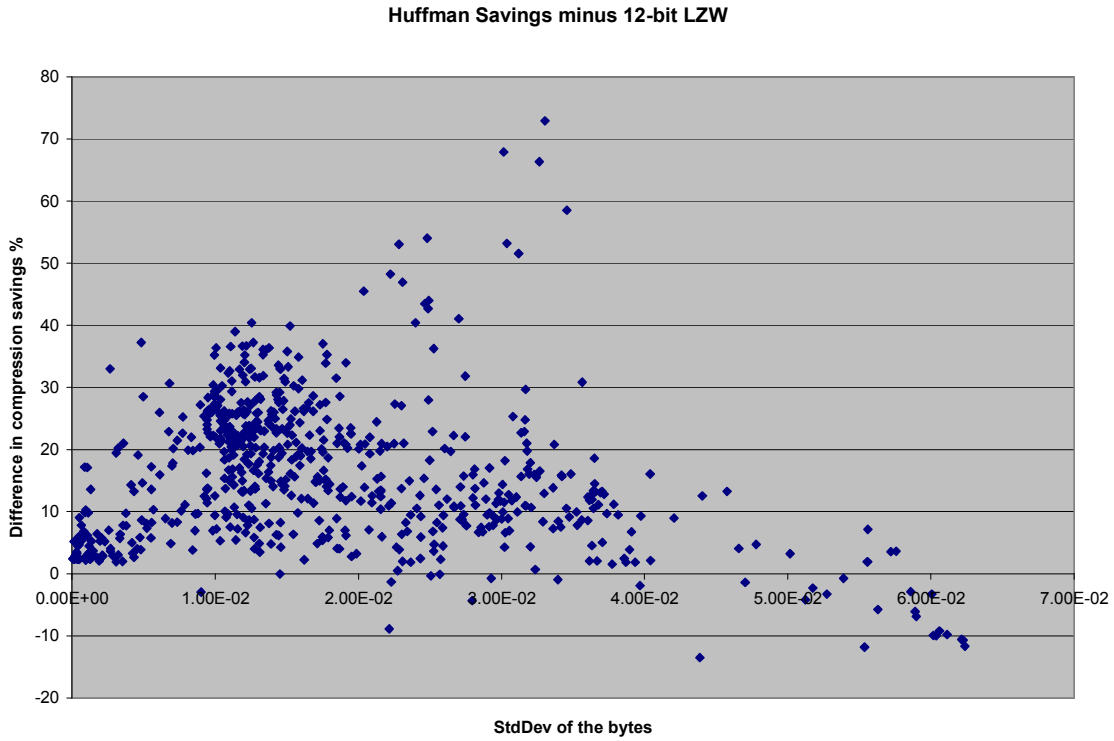


Figure 6: Graph showing the difference in savings percentage achieved for each data set through Huffman encoding and 12-bit LZW compression.

Figure 7 compares Huffman Encoding to the variable bit LZW algorithm. Points above the horizontal axis again represent greater savings when using Huffman. Using Equations 2 and 5, variable LZW should compress more at a standard deviation of the distribution of the bytes of over .0926. This is lower than the estimated equality point of Huffman and 12-bit LZW. This difference can be seen at the top end of the graph, where variable LZW seems to overtake Huffman before the estimated point. There is only one point with a standard deviation of the distribution of the bytes greater than .05 with a greater

compression savings using Huffman than variable LZW, and that point is less than .07% above being equal. Below .05, Huffman is generally better, as would be expected from the previous analysis. It is not, however, always better.

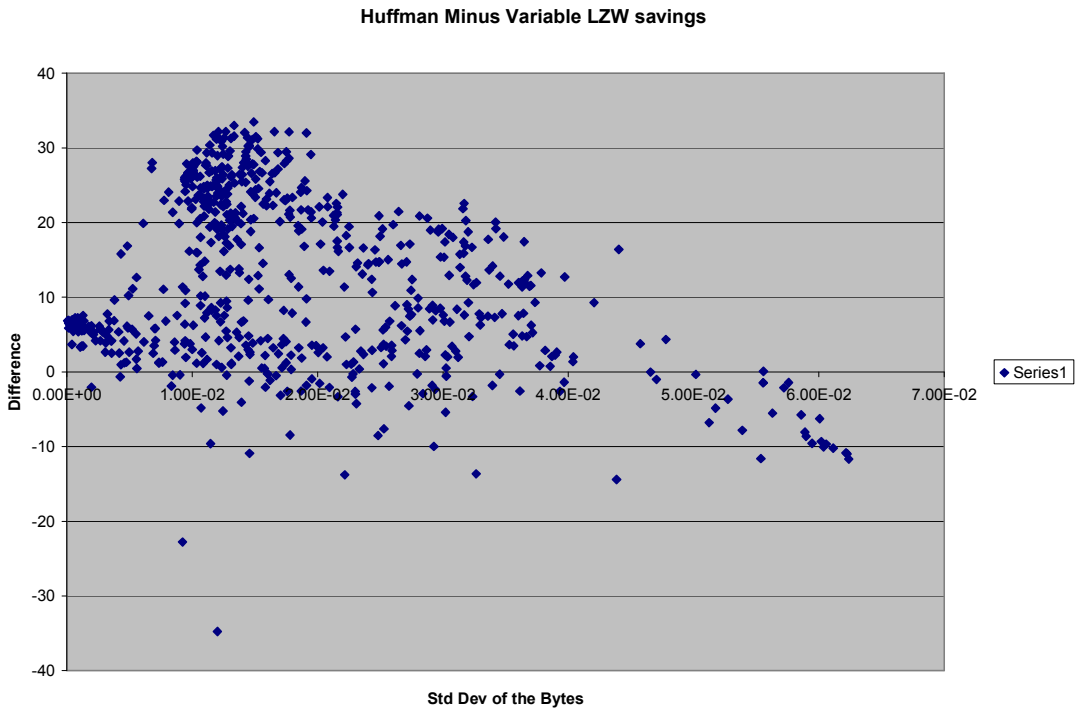


Figure 7: Graph showing the difference in savings percentage achieved for each data set through Huffman encoding and variable LZW compression.

4.2 12-bit LZW against variable bit LZW

Figure 8 compares 12-bit and variable bit LZW savings. Many of the points show that the two methods are often pretty close in their savings with variable LZW having the slight advantage, but there are a few significant points which point to variable bit having a larger advantage. 12-bit LZW never saves more than 11.1% over variable LZW, but there

are some significant savings in variable LZW over 12-bit LZW, including a handful of points with a savings of greater than 50% of the original file size.

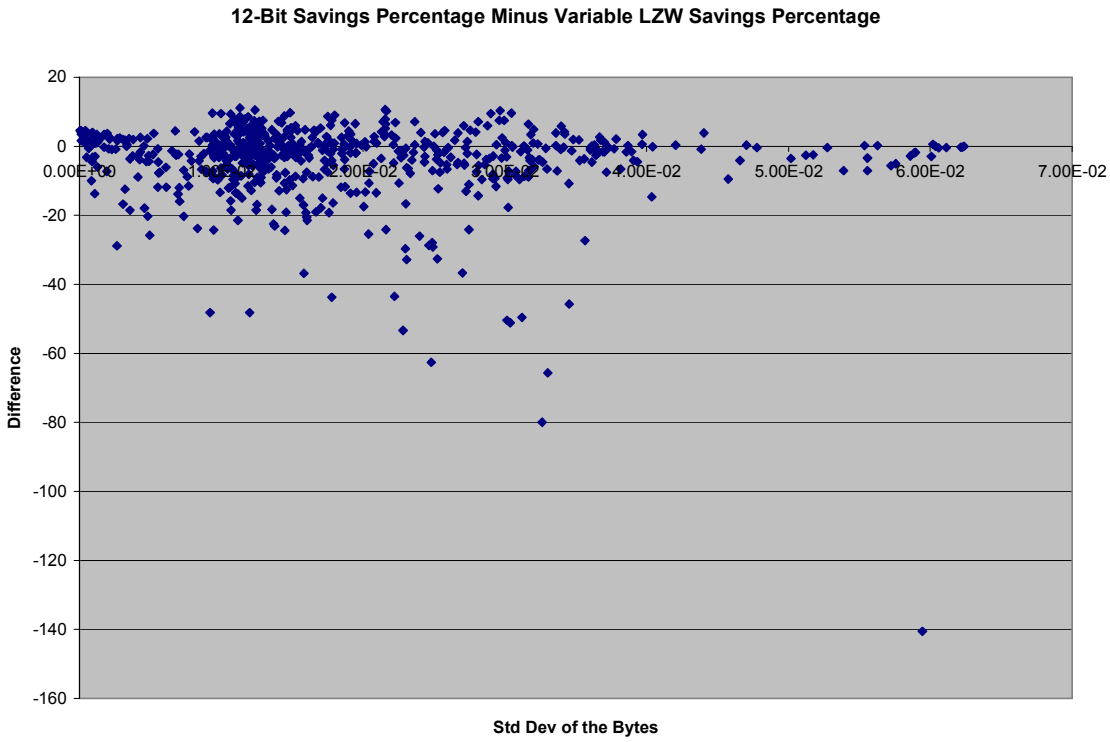


Figure 8: Graph showing the difference in savings percentage achieved for each data set through 12-bit and variable LZW compression.

Chapter 5 – Conclusions and Future Work

5.1 Conclusions

We made three assumptions that needed to be justified through the results. The first was the assumption of variety in the source data. The spread of the independent variable shows that there is a lot of variety in the data. The spread also justifies the assumption that files with the same extension have similar traits. When the files of like extensions were concatenated, trends in the distribution did not cancel each other and leave all of the data points in the center of the spread. The predictive ability of the trends also showed that there were some measurable trends within each data point. The concatenation of files did not dilute the predictive ability to a point where the trend was not obvious. There were not a large amount of outliers to show that a major dilution had occurred on many points. The existence of the trends also showed that the traits of the file were more important in finding a predictive relationship than the shape of the file. The similarities in the results of the two LZW algorithms also strengthened the justification of the second and third assumptions because the variable version relies less on the entire form of the file due to the clearing of the table every so often. That 12-bit followed similar trends showed that concatenating files did not severely alter most of the data points over considering subsections of those same files.

This project was successful at finding a single measure which predicts the compression savings of Huffman encoding and some varieties of LZW. The standard deviation of the

distribution of the bytes could explain more than 76% of the deviation in the savings compression for all the algorithms tested against it. This measure fulfills both of the requirements set out for the measure. It is faster and uses less space than multiple compressed files. The measure was calculated at the same time as creating the tables for the Huffman encoding in an effort to save time. The entire process took about 4 hours for the entire hard drive. This does not include the time to go back through the file and replace the bytes with their Huffman codes. In comparison, it took nearly four days to go through the same data using either LZW algorithm, and LZW is considered a high speed algorithm (Hayashi et al). The data structure required is an array of 256 number elements. These can be integers or floating points depending on the size of the file. If there are worries about underflow, multiple arrays can be used for keeping track, and the totals can be added at the end. Overall, it should take much less space than a compressed version of the file.

From the information gathered here, it would appear that Huffman encoding is usually better than 12-bit LZW. No clear measure was found for the cases in which the reverse is true. There were a few more points in which variable LZW fared better than Huffman, as can be seen on Figure 6, but they were mostly random in the distribution of the points. There was a point on the standard deviation of the distribution of the bytes scale at which a clear distinction could be made where variable LZW is better, but there were several points below that point where variable LZW is better which could not be predicted by the measure found in this project. Overall, Huffman encoding seems to usually be the best algorithm to use to get the largest percent savings for most data sets.

While the equations derived from the data using standard deviation of the distribution of the bytes were very predictive of the savings percentage, a direct comparison of the algorithms showed that the intersection of those equations was not the best predictive point for a change in which algorithm would achieve more savings. This was due to the overestimation of the predicted savings new the top end of the range for Huffman encoding. The measure was still decent at predicting when to use which algorithm, but the point at which the change in algorithms should be considered was considerably lower than the predicted point.

Among the LZW algorithms, there often wasn't too much of a difference. When 12-bit was compared to variable bit with a 12-bit limit, the algorithms often compared similarly, with variable bit having the slight advantage. This would indicate that using the front of the data set to estimate the distribution of the patterns in the file is a fairly decent practice, but there are some cases in which it falls short. By emptying the table and starting again, it is possible to allow the algorithm to form better to sections of the file instead of attempting to form to the file as a whole. There is a possible savings cost to using this approach, but it is never as substantial as the possible gain.

It is worth noting that the savings of the LZW algorithms were linearly related to the measure while the Huffman encoding savings were not. This could have something to do with the natural connection between logarithms and trees seen often in algorithm time analysis in computer science. No better explanation was found.

The compression savings of all the considered algorithms can be predicted by a measure which assesses the unevenness of the distribution of the bytes. This would seem to indicate that byte unevenness is an indicator of how well something can be compressed in general. By splitting the input into bytes, one could see if there is some byte which is underutilized. If there is, then there is an opening for some algorithm to map the data set into a more compressed form. This would seem to have a connection to the affect of entropy on the compression bounds that has been shown in other works.

5.2 Future Work

This project found a viable predictor for the compressibility of files, but there may be ways to cur down on the computational costs of finding. A properly chosen random sampling of the bytes should be able to return a value close to the actual standard deviation of the distribution of the bytes while only parsing a fraction of the file. Choosing the sample size and the algorithm for choosing that random sample of the bytes is left to future research.

The dictionary based algorithms have a variable that was not considered in this project. Both LZW versions considered here used a 12-bit dictionary. There is a potential for differences if smaller or larger dictionaries are used in either algorithm.

The linear trend for LZW against the standard deviation of the distribution of the bytes was obvious, and the logarithmic trend of Huffman against the same measure was clear enough. The reason for this difference is not, and was not studied here.

There was a potential advantage seen when using variable length LZW with a clearing of the table. It allowed the algorithm to form to subsets of the file instead of the file as a whole. It is possible that Huffman would benefit from a similar breaking apart of pieces of the data, but it would require more work to determine if and when this would be advantageous.

This research focused on two common types of compression. It may benefit from the expansion of the techniques to other algorithms, and potentially to the other two classes of algorithms.

The extent of the underutilized byte conjecture stated in the results also has potentially interesting lines of work. By breaking the file into strings of 8 bits, it appears to be possible to predict the compressibility of a file. It would be interesting to study if 8 bit strings are the most effective at predicting the compressibility or if another length would be better. It could also be interesting to study the extent of this relationship for other algorithms and inputs.

Chapter 6 – References

Acharya, Tinku and Sushmita Mitra. Data Mining: Multimedia, Soft Computing, and Bioinformatics. Hoboken: John Wiley and Sons, 2003.

Buchanan, W. J. *The Handbook of Data Communications and Networks.* 2 ed. I, Springer, 2004.

Crowcroft, Handley, and Wakeman. *Internet Multimedia*. CRC Press, 1999.

Gilchrist, Jeff. "Archive Comparison Test". http://compression.ca/act/. 9 August 2007. Last visited 13 February 2008.

Hayashi, S.; Kubo, J.-i.; Yamazato, T.; Sasase, I.; *A new source coding method based on LZW adopting the least recently used deletion heuristic*
Communications, Computers and Signal Processing, 1993., IEEE Pacific Rim Conference on
Volume 1, 19-21 May 1993 Page(s):190 - 193 vol.1
Digital Object Identifier 10.1109/PACRIM.1993.407191

"Lossless Data Compression Software Benchmarks". http://www.maximumcompression.com/index.html. 27 January, 2008. Last visited 13 February 2008.

Salomon, David. *Data Compression The Complete Reference.* 4 ed. London: Springer, 2007.